Efficient Algorithms for Envy-Free Stick Division With Fewest Cuts

Raphael Reitzig[*] Sebastian Wild[*]

April 13, 2015

Segal-Halevi, Hassidim, and Aumann [SHHA15] propose the problem of cutting sticks so that at least k sticks have equal length and no other stick is longer. This allows for an envy-free allocation of sticks to k players, one each. The resulting number of sticks should also be minimal.

We analyze the structure of this problem and devise a linear-time algorithm for it.

1. Introduction

We consider the following situation:

> *Imagine you and your family move to a new house. Naturally, each of your k children wants to have a room of their own, which is why you wisely opted for a house with a sufficient number of rooms. The sizes of the rooms are, however, not equal, and you anticipate that peace will not last long if any of the rascals finds out that their room is smaller than any of the others'.*
>
> *Removing walls is out of the question, but any room can (arbitrarily and iteratively) be divided into smaller rooms by installing drywalls. What is the minimal number of walls needed to obtain (at least) k equally sized largest rooms, (i. e., such that all other rooms are at most that large)?*

In this paper, we give efficient algorithms for solving this problem, i. e., for determining the optimal placement of walls. To the best of our knowledge, this is the first algorithmic treatment of this problem. Even though we do like the children's-rooms metaphor we want to remain consistent with existing descriptions of the problem and will therefore talk about *sticks* which can be cut (but not glued together), and refer to the problem as Envy-Free Stick Division – hence the title.

[*]Department of Computer Science, University of Kaiserslautern; {reitzig, wild}@cs.uni-kl.de

In the remainder of this first section, we give an overview over related problems and a roadmap of our contribution.

We then introduce notation and give a formal definition of Envy-Free Stick Division in Section 2. In Section 3 we develop the means to limit our search to finite sets of candidates. We follow up by developing first a linearithmic, then a linear time algorithm for solving Envy-Free Stick Division in Section 4. Finally, we propose smaller candidate sets Section 5. A short conclusion in Section 6 on the complexity of Envy-Free Stick Division completes the article.

We append a glossary of the notation we use in Appendix A for reference, and some lower bounds on the number of distinct candidates in Appendix B.

1.1. Other Optimization Goals

Whenever one defines a new problem, legitimate questions arise: Is the given definition natural? Do slight variations affect the properties of the problem?

We have already addressed the first question with the scenario right at the beginning of the article: when assigning rooms to children, each installed drywall incurs a cost which we try to minimize. The reader may or may not agree that this is a natural problem.

We now address the second question with a brief discussion of a few alternative definitions, none of which changes the nature of the problem.

To reiterate, we consider the problem of dividing resources (fairly), some of which may remain unallocated (as *waste*). Waste is bad, of course, so we want to avoid wasting too much. We can formulate this goal in different ways; one can seek to

(G1) minimize the number of necessary cuts (as we do),

(G2) minimize the number of waste pieces,

(G3) minimize the total amount of waste (here, the total length of all unallocated pieces), or

(G4) maximize the amount of resource each player gets (the length of the maximal pieces).

The first two objectives are discrete (counting things) whereas the latter two consider continuous quantities.

Obviously, (G3) and (G4) are dual formulations but lead to equivalent problems; the total waste is always the (constant) total length of all sticks minus $k \cdot l^\star$. Similarly, (G1) is dual to (G2); c cuts divide n sticks into $n + c$ pieces, and because exactly k of these are non-waste, the number of wasted pieces is $n + c - k$.

The canonical division induced by the largest feasible cut length l is also optimal w. r. t. the number of cuts needed; smaller cut lengths can only imply more cuts and larger cut lengths are not feasible. This result is one of the key insights towards algorithmic solutions of Envy-Free Stick Division so we state it formally in Corollary 3.2 (page 12). In the terms of above goals, this means that optimal solutions for (G3) and (G4) are also optimal for (G1) and (G2).

The converse is also true. Let $l^\star$ be the cut length of an optimal canonical division w. r. t. the number of needed cuts (goal (G1)). This division must divide (at least) one stick perfectly, that is into only maximal pieces; otherwise we could increase $l^\star$ a tiny bit until one stick is perfectly split, resulting in (at least) one cut less (as this stick does not produce a waste piece now). But this means that we cannot *increase* $l^\star$ by any (positive) amount without immediately losing (at least) one maximal piece; we formalize this fact in Lemma 3.1 (page 11). As the number of cuts grows with decreasing cut length, $l^\star$ is the largest cut length that yields at least k maximal pieces. Together it follows that $l^\star$ must be the largest feasible cut length overall and thus induces an optimal solution for goal (G4) (and therewith (G3)), too.

Therefore, all four objective functions we give above result in the same optimization problem, and the same algorithmic solutions apply. The reader may use any of the four formulations in case they do not agree with our choice of (G1).

1.2. Relation to Fair Division

This setting shares some features of fair resource allocation problems studied in economics, where a continuously divisible, homogeneous resource is to be allocated "fairly" to a number of competing players. See Brams and Taylor [BT96] for a treatise of this field. What exactly constitutes a "fair" division is subject to debate and several (potentially contradicting) notions of fairness appear in the literature:

- *proportional division* (or *simple fair division*) guarantees that each player gets a share that she values at least $1/k$;
- *envy-freeness* ensures that no player (strictly) prefers another player's share over her own;
- *equitable division* requires that the (relative) value each player assigns to her own share is the same for all players, so that everyone feels the same amount of "happiness".

This is just naming the most prominent ones. Our problem certainly is a search for an envy-free allocation of the available rooms/sticks.

The core assumption for classic fair division problems is the subjective theory of value, implying that each player has her own private (i. e., unknown) valuation of the resources at hand, which in general forbids an objectively fair division of the resources. Note that we do *not* assume subjective valuations in our problem; all the children value same rooms same: (linearly) by their size. This is what makes our problem easier than general fair resource allocation.

On the other hand, while each given room is assumed to be continuously divisible, existing walls (or cuts) constrain the set of possible allocations, and simply assigning everyone a (contiguous) $1/k$ share is not possible. In general there may not be an envy-free assignment then, at all, that is unless we allow wasting part of the resource. This is what makes our problem hard . . . and interesting.

While we do think that Envy-Free Stick Division is worth studying in its own right, it was initially motivated by a recent approach to envy-free cake cutting, i. e., the problem of finding an envy-free assignment of a freely divisible, but inhomogeneous resource (which means that valuations do not only depend on the size of the piece, but also on its position). Segal-Halevi, Hassidim, and Aumann [SHHA15] devise a finite protocol to find an envy-free division of a cake to k agents, where parts of the cake may remain unassigned (waste).

They use an algorithm for solving Envy-Free Stick Division as a subroutine in their protocol, which they call *Equalize*(k). In short, their protocol works as follows: The players cut pieces from the cake, one after another, only allowed to subdivide already produced pieces. After that, players each choose their favorite piece among the existing pieces in the *opposite* cutting order, i. e, the player who cut first is last to choose her piece of cake. During the cutting phase, agents produce a well-chosen number of pieces of the cake that they regard to be all of equal value (according to their personal valuation); all other pieces are made at most that large. The number of maximal pieces is chosen such that even after all players who precede the given player in "choosing order" have taken their favorite piece of cake, there is still at least one of the current player's maximal pieces left for her to pick, guaranteeing envy-freeness of the overall allocation.

1.3. Relation to Proportional Apportionment

Proportional apportionment as discussed by, e. g., Cheng and Eppstein [CE14], is the problem of assigning each party its "fair" share of a pool of *indivisible*, unit-value resource items (seats in parliament), so that the fraction of items assigned to a party resembles as closely as possible its (fractional, a priori known) *value* (the number of votes for a party); these values are the input.

Most methods used in real-word election systems use sequential assignment, assigning one seat at a time in a greedy fashion, where the current priority of each party depends on its initial vote percentage and the number of already assigned seats. Different systems differ in the function for computing these updated values, but all sensible ones seem to be of the "highest averages form": In each round they assign the next seat to a party that (currently) maximizes v_i/d_j (with ties broken arbitrarily), where v_i is the vote percentage of party i, j is the number of seats party i has already been assigned and $d_0, d_1, d_2, \ldots$ an increasing *divisor sequence* characterizing the method.

The arguably most natural choice is $d_j = j + 1$, yielding the highest average method of Jefferson, a. k. a. the greatest divisors method. Here v_i/d_j is the number of votes per seat that party i can exhibit, assuming it will now be given its $(j+1)$th seat. The party with the highest average number of votes per seat gets the next seat, which is fair in the sense that – correcting for the already fixed partial seat allocation – the voters value that party most. Cheng and Eppstein [CE14, Table 1] discuss other common choices for divisor sequences.

$d_0 = 0$ is allowed in which case v_i/d_0 is supposed to mean $v_i + M$ for a very large constant M (larger than the sum $\sum v_i$ of all values is sufficient). This ensures that before any party is assigned a second seat, all other parties must have one seat, which is a natural requirement for some allocations scenarios, e. g., the number of representatives for each state in a federal republic might be chosen to resemble population counts, but any state, regardless how small, should have at least one representative.

Note that even though the original description of the highest averages allocation procedure is an iterative process, it is actually a static problem: The averages v_i/d_j are strictly decreasing with j for any i, so the sequence of maximal averages in the assignment rounds is also decreasing. In fact, if we allocate a seat to a party with current average a in round r, then a must have been the rth largest element in the multiset of all ratios v_i/d_j for all parties i and numbers j. Moreover, if we know the value of the kth largest average $a^{(k)}$, we can determine for each party i how many seats it should receive by finding the largest j such that v_i/d_j is still larger than $a^{(k)}$. Cheng and Eppstein turn this idea into an efficient algorithm for solving apportionment problems.

Envy-Free Stick Division is in fact a proportional apportionment problem; we change our perspective and assign children to rooms (not vice versa!) so that each room gets its "fair share" of kids, namely proportional to its size. As children are (hopefully considered to be) indivisible, we can match size proportions only approximately. The increasing divisor sequence for Envy-Free Stick Division is $d_j = j + 1$, exactly the greatest divisors method discussed above.

Once we have determined the "share of children" $c_i \in \mathbb{N}_0$ of each room i of size L_i, we use the minimum of all L_i/c_i ratios as cut length $l^\star$ (ignoring rooms with $c_i = 0$ kids).

Therefore, Envy-Free Stick Division can be solved using Cheng and Eppstein's linear-time apportionment algorithm. We will, however, show in this article that we can – with a slightly closer look at the underlying multiset of averages v_i/d_j – devise an algorithm that is (in our opinion) both conceptually simpler and promises to be faster in practice.

1.4. Overview of this Article

In this section, we give an informal description of the steps that lead to our solution for Envy-Free Stick Division (see also Figure 1); formal definitions and proofs follow in the main part of the paper.

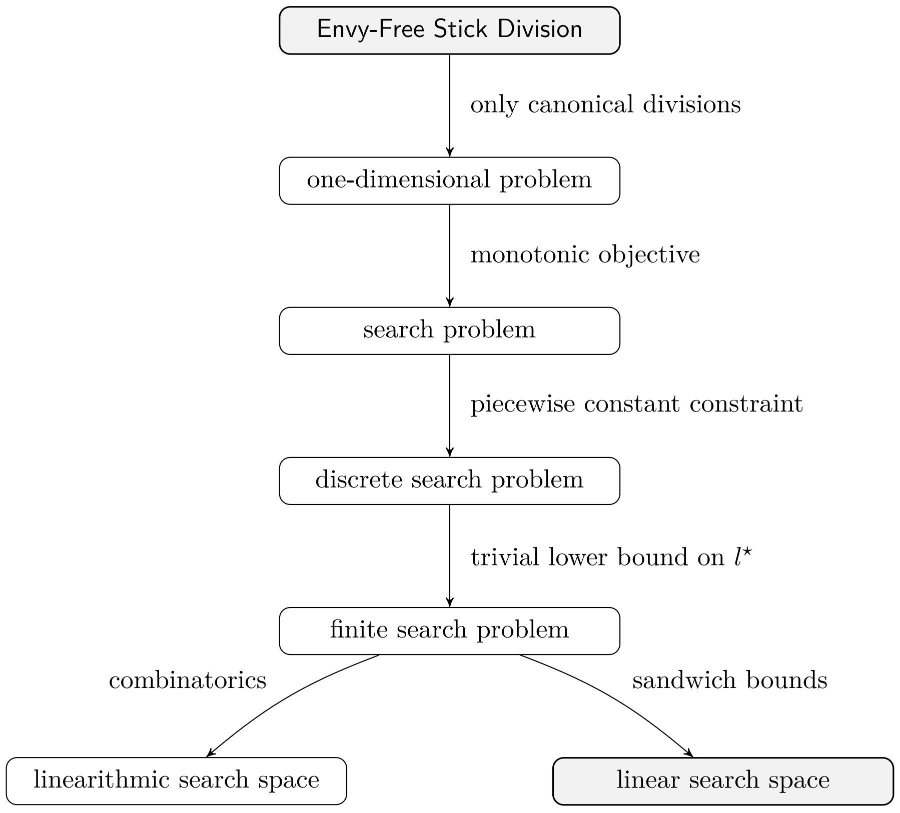

Figure 1: Schematic overview of the refinement steps that turn a seemingly hard problem into a tame task amenable to elementary yet efficient algorithmic solutions.

Without further restrictions, Envy-Free Stick Division is a non-linear continuous optimization problem that does not seem to fall into any of the usual categories of problems that are easy to solve. Any stick might be cut an arbitrary number of times at arbitrary lengths, so the space of possible divisions is huge.

The first step to tame the problem is to observe that most of these divisions cannot be optimal: Assuming we already know the size $l^\star$ of the k maximal pieces in an optimal division (the size of the rooms assigned to the kids), we can recover a *canonical* optimal division by simply cutting $l^\star$-sized pieces off of any stick longer than $l^\star$ until all sticks have length at most $l^\star$. Cutting a shorter piece only creates waste, cutting a larger piece always entails a second cut for that piece. We can thus identify a (candidate) cut length with its corresponding canonical division and so Envy-Free Stick Division reduces to finding the optimal cut length $l^\star$.

The second major simplification comes from the observation that for canonical divisions, the number of cuttings can only get larger when we decrease the cut length. (We cut each sticks into shorter pieces, this can only mean more cuts.) Stated differently, the objective function that we try to minimize is *monotonic* (in the cut length). This is a very fortunate situation since it allows a simple characterization of optimal solutions: $l^\star$ is the largest length, whose canonical division still contains (at least) k maximal pieces of equal size, transforming our optimization to a mere search problem for the point where cut lengths transition from feasible to infeasible solutions.

By similar arguments, also the number of equal sized maximal pieces (in the canonical division) for a cut length l does only increase when l is made smaller, so we can use *binary search* to find the length $l^\star$ where the number of maximal pieces first exceeds k. The search is still over a continuous region, though.

Next we note that both the objective and the feasibility function are piecewise constant with jumps only at lengths of the form $^{L_i}\!/_j$, where L_i is the length of an input stick and j is a natural number. Any (canonical) division for length l that does not cut any stick evenly into pieces of length l remains of same quality and cost if we change the l a very little. Moreover, any such division can obviously be improved by increasing the cut length, until we cut one stick L_i evenly, say into j pieces, as we then get the same number of maximal pieces with (at least) one less cutting. We can thus restrict our (binary) search for $l^\star$ to these jump points, making the problem discrete – but still infinite, as we do not yet have an upper bound on j.

We can, however, easily find lower bounds on $l^\star$ – or, equivalently, upper bounds on j – that render the search space finite. For example, we obviously never need to cut more than k pieces out of any single stick, in particular not the largest one. This elementary trick already reduces the search space to $O(n^2)$ candidates, where n is the number of sticks in the input.

We will then show how to obtain even smaller candidate sets by developing slightly cleverer upper and lower bounds for the number of maximal pieces (in the canonical divisions) for cut length l. The intuitive idea is as follows. If we had a single stick with the total length of all sticks, dividing it into k equal pieces would give us the ultimately efficient division without any waste. The corresponding "ultimate cut length" is of course easy to compute, but with pre-cut sticks, it will usually not be feasible.

However, we can *bound* how much we lose w. r. t. the ultimate division: each input stick contributes (at most) one piece of waste. Hence, the optimal cut length cannot be "too far" away from the ultimate cut length. With a little diligence (see Section 5.1), we can derive an interval for the optimal cut length from these observations and show that the number of jumps in this interval, i. e., the number of cut lengths to check, remains linear in n. For $k \le n$, we can in fact get an $O(k)$ bound by first removing sticks shorter than the kth largest one.

The discussion above takes the point of view of mathematical optimization, describing how to reduce the number of candidate cut lengths we have to check; we are still one step away from turning this into an actual, executable algorithm. After reducing the problem to a finite search problem, binary search naturally comes to mind; we work out the details in Section 4. However, *sorting* the candidate set and *checking feasibility* of candidates dominate the runtime of this binary-search-based algorithm – this is unsatisfactory.

As hinted at in Section 1.3, it is possible to determine $l^\star$ more directly, namely as a specific order statistic of the candidate set. From the point of view of objective and feasibility functions, this trick works because both functions essentially *count* the number of unit jumps (i. e. occurrences of ${}^{L_i}/_{j}$) at points larger than the given length. This approach yields a simple linear-time algorithm based on rank selection; we describe it in detail in Section 4.1.

2. Problem Definition

We will mostly use the usual zoo of mathematical notation (as used in theoretical computer science, that is). Since they are less often used, let us quickly introduce notation for *multisets*, though. For some set X, denote a multiset over X by

$$\mathbf{A} = \{x_1, x_2, \dots\}$$

with $x_i \in X$ for all $i \in [1..|\mathbf{A}|]$. Note the bold letter; we will use these for multisets, and regular letters for sets. Furthermore, denote the *multiplicity* of some $x \in X$ in $\mathbf{A}$ as $\mathbf{A}(x)$; in particular,

$$|\mathbf{A}| = \sum_{x \in X} \mathbf{A}(x).$$

When we use a multiset as operator range, we want to consider every occurrence of $x \in \mathbf{A}$; for example,

$$\sum_{x \in \mathbf{A}} f(x) = \sum_{i \in [1..|\mathbf{A}|]} f(x_i) = \sum_{x \in X} \mathbf{A}(x) \cdot f(x).$$

As for multiset operations, we use *multiset union* $\uplus$ that adds up cardinalities; that is, if $\mathbf{C} = \mathbf{A} \uplus \mathbf{B}$ then $\mathbf{C}(x) = \mathbf{A}(x) + \mathbf{B}(x)$ for all $x \in X$. *Multiset difference* works in the reverse way; if $\mathbf{C} = \mathbf{A} \setminus \mathbf{B}$ then $\mathbf{C}(x) = \max\{0, \mathbf{A}(x) - \mathbf{B}(x)\}$ for all $x \in X$.

Intersection with a set $B \subseteq X$ is to be read as natural extension for the usual set intersection; that is, if $\mathbf{C} = \mathbf{A} \cap B$ then $\mathbf{C}(x) = \mathbf{A}(x) \cdot B(x)$ for all $x \in X$ (keep in mind that $B(x) \in \{0, 1\}$).

Now for problem-specific notation. We call any length $L \in \mathbb{Q}$ a *stick*. *Cutting* L with length $l > 0$ creates two pieces with lengths l and $L - l$ respectively. By iteratively cutting sticks and pieces thereof into smaller pieces, we can transform a set of sticks into a set of pieces.

We define the following simple problem for fixing notation.

Problem 1: Envy-Free Fixed-Length Stick Division

> **Input:** Multiset $\mathbf{L} = \{L_1, \dots, L_n\}$ of sticks with lengths $L_i \in \mathbb{Q}_{>0}$, target number $k \in \mathbb{N}_{>0}$ and cut length $l \in \mathbb{Q}_{>0}$.
>
> **Output:** The (minimal) number of cuts necessary for cutting the input sticks into sticks $L'_1, \dots, L'_{n'} \in \mathbb{Q}_{>0}$ so that
>
> i) (at least) k pieces have length l, i.e. $|\{i \mid L'_i = l\}| \geq k$,
>
> ii) and no piece is longer than l, i.e. $L'_i \leq l$ for all i.

The solution is elementary; we give it after introducing formal notation.

We denote by $m(L, l)$ the number of stick pieces of length l – we will also call these *maximal* pieces – we can get when we cut stick L into pieces no longer than l. This is to mean that you first cut L into two pieces, then possibly further cut those pieces and so on, until all pieces have length at most l. Obviously, the best thing to do is to only ever cut with length l. We thus have

$$m(L, l) = \left\lfloor \frac{L}{l} \right\rfloor.$$

Because we may also produce one shorter piece, the total number of pieces we obtain by this process is given by

$$p(L, l) = \left\lceil \frac{L}{l} \right\rceil,$$

and

$$c(L, l) = \left\lceil \frac{L}{l} \right\rceil - 1$$

denotes the number of cuts we perform.

We extend this notation to multisets of sticks, that is

$$\begin{aligned} m(\mathbf{L}, l) \; &:= \; \sum_{L \in \mathbf{L}} m(L, l) \; = \; \sum_{L \in \mathbf{L}} \left\lfloor \frac{L}{l} \right\rfloor \text{ and} \\ c(\mathbf{L}, l) \; &:= \; \sum_{L \in \mathbf{L}} c(L_i, l) \; = \; \sum_{L \in \mathbf{L}} \left\lceil \frac{L}{l} - 1 \right\rceil. \end{aligned}$$

See Figure 2 for a small example.

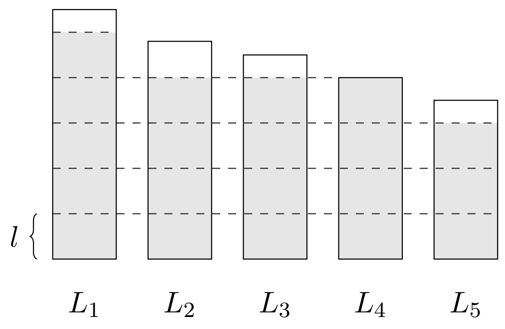

Figure 2: Sticks $\mathbf{L} = \{L_1, \dots, L_5\}$ cut with some length l. Note how $m(\mathbf{L}, l) = 20$ and $c(\mathbf{L}, l) = 19$. There are four non-maximal pieces.

Using this notation, the conditions of Envy-Free Fixed-Length Stick Division translate into checking whether $m(\mathbf{L}, l) \geq k$ for cut length l; we call such l *feasible* cut lengths (for $\mathbf{L}$ and k). We define the following predicate as a shorthand:

$$\text{Feasible}(\mathbf{L}, k, l) \ := \begin{cases} 1, & m(\mathbf{L}, l) \geq k; \\ 0, & \text{otherwise.} \end{cases}$$

Now we can give a concise algorithm for solving Envy-Free Fixed-Length Stick Division.

Algorithm 1: CanonicalCutting($\mathbf{L}, k, l$) :

```
1. If Feasible($\mathbf{L}, k, l$):
    1.1. Answer $c(\mathbf{L}, l)$.
2. Otherwise:
    2.1. Answer $\infty$ (i. e. “not possible”).
```

Assuming the unit-cost RAM model – which we will do in this article – the runtime of CanonicalCutting is clearly in $O(n)$; evaluation of Feasible and c in time $O(n)$ each dominates. We will see later that a better bound is $\Theta(\min(k, n))$ (cf. Lemma 3.3).

Of course, different cut lengths l cause different numbers of cuts. We want to find an *optimal* cut length, that is a length $l^\star$ which *minimizes* the number of cuts necessary to fulfill conditions i) and ii) of Envy-Free Fixed-Length Stick Division. We formalize this as follows.

Problem 2: Envy-Free Stick Division

Input: Multiset $\mathbf{L} = \{L_1, \dots, L_n\}$ of sticks with lengths $L_i \in \mathbb{Q}_{>0}$ and target number $k \in \mathbb{N}_{>0}$.

Output: A (feasible) cut length $l^\star \in \mathbb{Q}_{>0}$ which minimizes the result of Envy-Free Fixed-Length Stick Division for $\mathbf{L}$, k and $l^\star$.

We observe that the problem is not as easy as picking the smallest L_i, cutting the longest stick into k pieces, or using the kth longest stick (if $k \leq n$). Consider the following, admittedly artificial example which debunks all such attempts.

Example 2.1: Let

$$\mathbf{L} = \{mx, (m-1)x + 1, (m-2) \cdot x + 2, \dots, {}^{m}/_{2} \cdot x + {}^{m}/_{2}, x-1, x-1, \dots\}$$

for a total of $n = m^2$ elements and $k = {}^{3}/_{8} \cdot m^2 + {}^{3}/_{4} \cdot m$, where $m \in 4\mathbb{N}_{>0}$ and $x > {}^{m}/_{2}$.

Note that $l^\star = x$, that is in particular

- $l^\star \neq L_i$ and
- $l^\star \neq {}^{L_i}/_{k}$

for all $i \in [1..n]$. In fact, by controlling x we get an (all but) arbitrary fraction of an L_i for $l^\star$. It is possible to extend the example so that "mx" – the stick whose fraction is optimal – has (almost) arbitrary index i, too.

As running example we will use $(\mathbf{L}_{\text{ex}}, k)$ as defined by $m = 4$ and $x = 2$, that is

- $\mathbf{L}_{\text{ex}} = \{8, 7, 6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}$ and
- $k = 9$.

Note that $l^\star = 2$ and $m(\mathbf{L}_{\text{ex}}, l^\star) = 10 > k$ here. See Figure 3 for a plot of $m(\mathbf{L}_{\text{ex}}, l)$.

3. Exploiting Structure

For ease of notation, we will from now on assume arbitrary but fixed input $(\mathbf{L}, k)$ be given implicitly. In particular, we will use $m(l)$ as short form of $m(\mathbf{L}, l)$, and similar for c and Feasible.

At first, we observe that both constraint and objective function of Envy-Free Stick Division belong to a specific, simple class of functions.

Lemma 3.1: *Functions m and c are non-increasing, piecewise-constant functions in l with jump discontinuities of (only) the form ${}^{L_i}/_{j}$ for $i \in [1..n]$ and $j \in \mathbb{N}_{>0}$.*

Furthermore, m is left- and c is right-continuous.

Proof: The functions are given as finite sums of terms that are either of the form $\lfloor \frac{L}{l} \rfloor$ or $\lceil \frac{L}{l} - 1 \rceil$. Hence, all summands are piecewise constant and never increase with growing l. Thus, the sum is also a non-increasing piecewise-constant function.

The form ${}^{L_i}/_{j}$ of the jump discontinuities is apparent for each summand individually, and they carry over to the sums by monotonicity.

The missing continuity properties of m resp. c follow from right-continuity of $\lfloor \cdot \rfloor$ resp. left-continuity of $\lceil \cdot \rceil$; the direction gets turned around because we consider l^{-1} but other than that arithmetic operations maintain continuity. $\square$

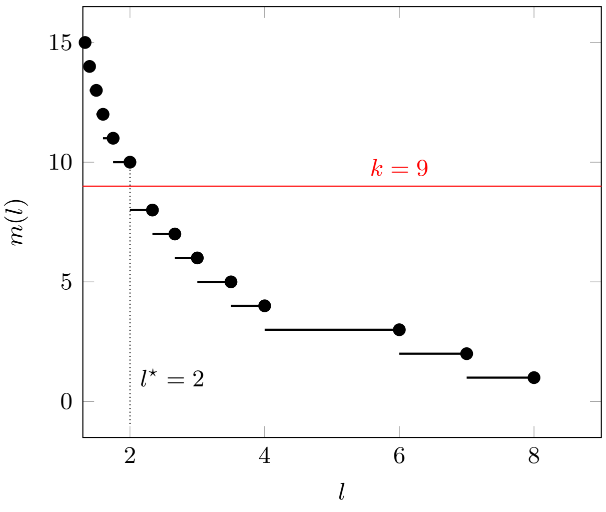

Figure 3: The number of maximal pieces $m(\mathbf{L}_{\text{ex}}, l)$ in cut length l for $\mathbf{L}_{\text{ex}}$ as defined in Example 2.1. The filled circles indicate the value of $m(\mathbf{L}_{\text{ex}}, l)$ at the jump discontinuities.

See Figure 3 for an illustrating plot.

Knowing this, we immediately get lots of structure in our solution space which we will utilize thoroughly.

Corollary 3.2: $l^\star = \max\{l \in \mathbb{Q}_{>0} \mid \text{Feasible}(l)\}$. □

Note in particular that the maximum exists because Feasible is left-continuous.

This already tells us that any feasible length gives a lower bound on $l^\star$. One particular simple case is $k < n$ since then the kth largest stick is always feasible. This allows us to get rid of all properly smaller input sticks, too, since they are certainly waste when cutting with any optimal length. As a consequence, having *any* non-trivial lower bound on $l^\star$ already speeds up our search by ways of speeding up feasibility checks.

Lemma 3.3: *Let $L \in \mathbb{Q}_{\geq 0}$ fixed and denote with*

$$I_{>L} := \{i \in [1..n] \mid L_i > L\}$$

the (index) set of all sticks in $\mathbf{L}$ that are longer than L. Then,

$$m(l) = \sum_{i \in I_{>L}} m(L_i, l)$$

for all $l > L$.

Proof: Clearly, all summands $\lfloor L_i/l \rfloor$ in the definition of $m(l)$ are zero for $L_i \leq L < l$. □

As a direct consequence, we can push the time for checking feasibility of a candidate solution from being proportional to n down to being proportional to the number of L_i larger than a lower bound L on the optimal length; we simply preprocess $\mathbf{L}_{>L}$ in time $\Theta(n)$. Since it is easy to find an L_i that can serve as L – e.g. any one that is shorter than any known feasible solution – we will make use of this in the definition of our set of candidate cut lengths.

In addition, the special shape of c and Feasible comes in handy. Recall that both functions are step functions with (potential) jump discontinuities at lengths of the form $l = {}^{L_i}\!/_{j}$ (cf. Lemma 3.1). We will show that we can restrict our search for optimal cut lengths to these values, and how to do away with many of them for efficiency.

Combining the two ideas, we will consider candidate multisets of the following form.

Definition 3.4: *We define the candidate multiset(s)*

$$\mathcal{C}(I, f_l, f_u) \;:=\; \biguplus_{i \in I} \left\{ \frac{L_i}{j} \;\middle|\; f_l(i) \leq j \leq f_u(i) \right\}$$

dependent on index set $I \subseteq [1..n]$ and functions $f_l : I \to \mathbb{N}$ and $f_u : I \to \mathbb{N} \cup \{\infty\}$ which bound the denominator from below and above, respectively; either may implicitly depend on $\mathbf{L}$ and/or k.

Note that $|\mathcal{C}| = \sum_{i \in I}[f_u(i) - f_l(i) + 1]$. We denote the multiset of all candidates by $\mathcal{C}_{\text{all}} := \mathcal{C}([1..n], 1, \infty)$.

First, let us note that this definition covers the optimal solution as long as upper and lower bounds are chosen appropriately.

Lemma 3.5: *There is an optimal solution on a jump discontinuity of m, i.e. $l^\star \in \mathcal{C}_{\text{all}}$.*

Proof: From its definition, we know that Feasible has exactly one jump discontinuity, and from Lemma 3.1 (via m) we know that it is one of the ${}^{L_i}\!/_{j}$. By Corollary 3.2 and left-continuity of Feasible (again via m) we know that this is indeed our solution $l^\star$. $\square$

Of course, our all-encompassing candidate multiset $\mathcal{C}_{\text{all}}$ is infinite (as is the corresponding set) and does hence not lend itself to a simple search. But there is hope: we already know that $l^\star \geq l$ for any feasible l which immediately implies finite (if maybe super-polynomial) bounds on j (if we have such l). We will now show how to restrict the set of candidates via suitable index sets I and bounding functions f_l and f_u so that we can efficiently search for $l^\star$. We have to be careful not to inadvertently remove $l^\star$ by choosing bad bounding functions.

Lemma 3.6: *Let $I \subseteq [1..n]$ and $f_l, f_u : I \to \mathbb{N}$ so that*

i) $f_l(i) = 1$ or ${}^{L_i}\!/_{(f_l(i)-1)}$ is infeasible, and *ii)* ${}^{L_i}\!/_{f_u(i)}$ *is feasible,*

for all $i \in I$*, and*

iii) $L_{i'}$ *is suboptimal (i. e.* $L_{i'}$ *is feasible, but not optimal)*

for all $i' \in [1..n] \setminus I$*. Then,*

$$l^\star \in \boldsymbol{\mathcal{C}}(I, f_l, f_u).$$

Proof: We argue that $\boldsymbol{\mathcal{C}}_{\text{all}} \setminus \boldsymbol{\mathcal{C}}(I, f_l, f_u)$ does *not* contain the optimal solution $l^\star$; the claim then follows with Lemma 3.5.

Let $i \in [1..n]$ and $j \in [1..\infty]$ be arbitrary but fixed. We investigate three cases for why length ${}^{L_i}\!/_j$ may not be included in $\boldsymbol{\mathcal{C}}(I, f_l, f_u)$.

$\boldsymbol{i \notin I}$**:** Candidate ${}^{L_i}\!/_j$ is suboptimal by Corollary 3.2 because ${}^{L_i}\!/_j \leq L_i$ and L_i itself is already suboptimal by iii).

$\boldsymbol{j < f_l(i)}$**:** In this case, we must have $f_l(i) > 1$, so ${}^{L_i}\!/_{(f_l(i)-1)}$ is infeasible by i). Clearly, ${}^{L_i}\!/_j > {}^{L_i}\!/_{f_l(i)}$, so ${}^{L_i}\!/_j \geq {}^{L_i}\!/_{(f_l(i)-1)}$ and we get by monotonicity of Feasible (cf. Lemma 3.1 via m) that ${}^{L_i}\!/_j$ is infeasible, as well.

$\boldsymbol{j > f_u(i)}$**:** Clearly, ${}^{L_i}\!/_j < {}^{L_i}\!/_{f_u(i)}$, where the latter is already feasible by ii). So, again by Corollary 3.2, ${}^{L_i}\!/_j$ is suboptimal.

Thus, we have shown that every candidate length ${}^{L_i}\!/_j$ given by $(i, j) \in I \times [1..\infty]$ is either in $\boldsymbol{\mathcal{C}}(I, f_l, f_u)$ or, failing that, infeasible or suboptimal. □

We will call triples (I, f_l, f_u) of index set and bounding functions that fulfill Lemma 3.6 *admissible restrictions* (for $\mathbf{L}$ and k). We say that $\boldsymbol{\mathcal{C}}(I, f_l, f_u)$ is admissible if (I, f_l, f_u) is an admissible restriction.

We will restrict ourselves for the remainder of this article to index sets I_{co} that contain indices of lengths that are larger than the n'th largest[1] length $L^{(n')}$ in $\mathbf{L}$, for $n' = \min(k, n+1)$. This corresponds to working with $I_{>L^{(n')}}$ as defined in Lemma 3.3. We will have to show that such index sets are indeed admissible (alongside suitable bounding functions); intuitively, if $k \leq n$ then $L^{(k)}$ is always feasible, and otherwise we have to work with all input lengths. We fix this convention for clarity and notational ease.

Definition 3.7: *Define cut-off length* L_{co} *by*

$$L_{\text{co}} := \begin{cases} L^{(k)}, & k \leq n; \\ 0, & k > n, \end{cases}$$

[1]We borrow from the common notation $S_{(k)}$ for the kth smallest element of sequence S.

and index set $I_{\text{co}} \subseteq [1..n]$ as

$$I_{\text{co}} := \begin{cases} I_{>L_{\text{co}}}, & L_{\text{co}} \text{ not optimal;} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Note that $I_{\text{co}} = [1..n]$ if $k > n$.

We will later see that we never invoke the undefined case as we already have $l^\star = L^{(k)}$ then.

In order to illustrate that we have found a useful criterion for admissible bounds, let us investigate shortly an admittedly rather obvious choice of bounding functions. We use the null-bound $f_l = i \mapsto 1$ and $f_u = i \mapsto k$; an optimal solution does not cut more than k (equal-sized) pieces out of any one stick. The restriction $([1..n], 1, k)$ is clearly admissible; in particular, every ${}^{L_i}/_k$ is feasible.

Example 2.1 Continued: For $\mathbf{L}_{\text{ex}}$ and $k = 9$, we get

$$\mathcal{C}(I_{\text{co}}, 1, k) = \left\{ \frac{8}{1}, \frac{8}{2}, \frac{8}{3}, \frac{8}{4}, \frac{8}{5}, \frac{8}{6}, \frac{8}{7}, \frac{8}{8}, \frac{8}{9}, \frac{7}{1}, \frac{7}{2}, \frac{7}{3}, \frac{7}{4}, \frac{7}{5}, \frac{7}{6}, \frac{7}{7}, \frac{7}{8}, \frac{7}{9}, \right.$$
$$\left. \frac{6}{1}, \frac{6}{2}, \frac{6}{3}, \frac{6}{4}, \frac{6}{5}, \frac{6}{6}, \frac{6}{7}, \frac{6}{8}, \frac{6}{9}, \frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9} \right\},$$

that is 36 candidates. Note that there are four duplicates, so there are 32 distinct candidates.

We give a full proof of admissibility and worst-case size here; it is illustrative even if simple because later proofs will follow the same structure.

Lemma 3.8: *If $L_{\text{co}} \neq l^\star$, then $\mathcal{C}(I_{\text{co}}, 1, k)$ is admissible.*
Furthermore, $|\mathcal{C}(I_{\text{co}}, 1, k)| = k \cdot \min(k-1, n) \in \Theta(k \cdot \min(k, n))$ in the worst case.

Proof: First, we show that $(I_{\text{co}}, 1, k)$ is an admissible restriction (cf. Lemma 3.6).

ad i): Since $f_l(i) = 1$ for all i, this checks out.

ad ii): Clearly, $m({}^{L_i}/_k) \geq k$ just from the contribution of summand $\lfloor {}^{L_i}/_l \rfloor$.

ad iii): We distinguish the two cases of L_{co} (cf. Definition 3.7).

- If $k > n$ then $I_{\text{co}} = [1..n]$ which trivially fulfills iii).
- In the other case, $k \leq n$ and L_{co} is not optimal by assumption. Then $I_{\text{co}} = I_{>L_{\text{co}}}$; therefore $L_{i'} \leq L_{\text{co}}$ for any $i' \notin I_{\text{co}}$ and Lemma 3.1 implies that $L_{i'}$ is not optimal as well.

This concludes the proof of the first claim.

As for the number of candidates, note that clearly $|\mathcal{C}(I_{\text{co}}, 1, k)| = \sum_{i \in I_{\text{co}}} k = |I_{\text{co}}| \cdot k$; the claim follows with $|I_{\text{co}}| = \min(k-1, n)$ in the case that the L_i are pairwise distinct (cf. Definition 3.7). □

Since we now know that we have to search only a finite domain for $l^\star$, we can start thinking about effective and even efficient algorithms.

4. Algorithms

Just from the discussion above, a fairly elementary algorithm presents itself: first cut the input down to the lengths given by I_{co} (cf. Definition 3.7), then use binary search on the candidate set w. r. t. Feasible. This works because Feasible is non-increasing (cf. Corollary 3.2 and Lemma 3.1).

Algorithm 2: $\textsc{SearchLstar}\langle f_l, f_u\rangle(\mathbf{L}, k)$:

```
1. Compute $n' := \min(k, n+1)$.
2. If $n' \leq n$:
    2.1. Determine $L_{\text{co}} := L^{(n')}$, i. e. the $n'$th largest length.
    2.2. If $L_{\text{co}}$ is optimal, answer $l^\star = L_{\text{co}}$ (and terminate).
2'. Otherwise (i. e. $n' > n$):
    2.4. Set $L_{\text{co}} := 0$.
3. Assemble $I_{\text{co}} := I_{>L_{\text{co}}}$.
4. Compute $\mathcal{C} := \mathcal{C}(I_{\text{co}}, f_l, f_u)$ as sorted array.
5. Find $l^\star$ by binary search on $\mathcal{C}$ w. r. t. Feasible.
6. Answer $l^\star$.
```

For completeness we specify that $f_l, f_u : I_{\text{co}} \to \mathbb{N}$.

Theorem 4.1:
Let $(I_{\text{co}}, f_l, f_u)$ be an admissible restriction where f_l and f_u can be evaluated in time $O(1)$.

Then, algorithm $\textsc{SearchLstar}\langle f_l, f_u\rangle$ solves Envy-Free Stick Division in (worst-case) time

$$T(n,k) \in \Theta(n + |\mathcal{C}| \log |\mathcal{C}|)$$

and space

$$S(n,k) \in \Theta(n + |\mathcal{C}|).$$

Proof: We deal with the three claims separately.

Correctness follows immediately from Lemma 3.6 and Lemma 3.1 resp. Corollary 3.2. Note in particular that SEARCHLSTAR does indeed compute I_{co} as defined in Definition 3.7, and the undefined case is never reached.

Runtime: Since the algorithm contains neither loops nor recursion (at the top level) we can analyze every step on itself.

Steps 1, 2.4.: These clearly take time $O(1)$.

Step 2.1.: There are well-known algorithms that perform selection in worst-case time $\Theta(n)$.

Step 2.2.: Testing L_{co} for optimality is as easy as computing Feasible(L_{co}) and counting the number a of integral $^{L_i}/_{L_{\text{co}}}$ in $m(L_{\text{co}})$. If Feasible(L_{co}) (i. e., $m(L_{\text{co}}) \geq k$) and $m(L_{\text{co}}) - a < k$, then L_{co} is the jump discontinuity of Feasible and L_{co} is optimal; otherwise it is not.

Thus, this step takes time $\Theta(n)$.

Step 3: This can be implemented by a simple iteration over $[1..n]$ with a constant-time length check per entry, hence in time $\Theta(n)$.

Steps 3 I_{co} can be assembled by one traversal over $\mathbf{L}$ and stored as simple linked list in (worst-case) time $\Theta(n)$.

Step 4: By Definition 3.4 we have $|\mathcal{C}|$ many candidates. Sorting these takes time $\Theta(|\mathcal{C}| \log |\mathcal{C}|)$ using e. g. Heapsort.

Step 5: The binary search clearly takes at most $\lfloor \log_2 |\mathcal{C}| + 1 \rfloor$ steps. In each step, we evaluate Feasible in time

- $\Theta(|I_{\text{co}}|)$ for all candidates $l > L_{\text{co}}$ using Lemma 3.3, and
- $O(1)$ for $l \leq L_{\text{co}}$ since we already know from feasibility of L_{co} via Lemma 3.1 that these are feasible, too.

Therefore, this step needs time $\Theta(|I_{\text{co}}| \log |\mathcal{C}|)$ time in total.

It is easy to see that admissible bounds always fulfill $f_u(i) \geq f_l(i)$ for all $i \in I_{\text{co}}$. Therefore, $|I_{\text{co}}| \leq |\mathcal{C}|$ so the runtime of this step is dominated by step 4.

Space: The algorithm stores $\mathbf{L}$ of size $\Theta(n)$, plus maybe a copy for selection and partitioning (depends the actual algorithm used). Step 4 then creates a $\Theta(|\mathcal{C}|)$-large representation of the candidate set. Both step 3 and 5 can be implemented iteratively, and a potential recursion depth (and therefore stack size) in step 2.1. is bounded from above by its runtime $O(n)$. A few additional auxiliary variables require only constant amount of memory. □

For practical purposes, eliminating duplicates in Step 4 is virtually free and can speed up the subsequent search. In the worst case, however, we save at most a constant factor with the bounding functions we consider (see Appendix B), so we decided to stick to the clearer presentation using multisets (instead of candidate *sets*).

4.1. Knowing Beats Searching

We have seen that the runtime of algorithm SEARCHLSTAR is dominated by *sorting* the candidate set. This is necessary for facilitating binary search; but do we *have* to search? As it turns out, a slightly different point of view on the problem allows us to work with the unsorted candidate multiset and we can save a factor $\log |\boldsymbol{\mathcal{C}}|$.

The main observation is that m increases its value by one at every jump discontinuity (for each L_i/j that has that same value). So, knowing $m(l)$ for any candidate length l, we know exactly how many candidates (counting duplicates) we have to move to get to the jump of Feasible. Therefore, we can make do with *selecting* the solution from our candidate set instead of searching through it.

The following lemma states the simple observation that $m(L, l)$ is intimately related to the "position" of l in the decreasingly sorted candidate multiset for L.

Lemma 4.2: *For all $L, l \in \mathbb{Q}_{>0}$,*

$$m(L, l) \;=\; \bigl|\{L/j \mid j \in \mathbb{N} \wedge L/j \geq l\}\bigr|.$$

Proof: The right-hand side equals the largest integer $j \in \mathbb{N}_0$ for which $L/j \geq l$, i. e. $j \leq L/l$, which is by definition $\lfloor L/l \rfloor = m(L, l)$. Note that this argument extends to the case $L < l$ by formally setting $L/0 = \infty \geq l$. □

Since we consider multisets, we can lift this property to $m(l)$:

Corollary 4.3: *For all $l \in \mathbb{Q}_{>0}$,*

$$m(l) \;=\; \sum_{i=1}^{n} \bigl|\{L_i/j \mid j \in \mathbb{N} \wedge L_i/j \geq l\}\bigr| \;=\; \bigl|\boldsymbol{\mathcal{C}}_{\text{all}} \cap [l, \infty)\bigr|.$$ □

In other words, $m(l)$ is the number of occurrences of candidates that are at least l. We can use this to transform our search problem (cf. Corollary 3.2) into a selection problem.

Lemma 4.4: $l^\star = \boldsymbol{\mathcal{C}}_{\text{all}}^{(k)}$.

Proof: Denote with $\mathcal{C}_{\text{all}}$ the *set* of all candidates, that is $l \in \mathcal{C}_{\text{all}} \iff \boldsymbol{\mathcal{C}}_{\text{all}}(l) > 0$. We can thus write the statement of Corollary 4.3 as

$$m(l) = \sum_{\substack{l' \in \mathcal{C}_{\text{all}} \\ l' \geq l}} \boldsymbol{\mathcal{C}}_{\text{all}}(l'). \tag{1}$$

i	1	2	3	4	5	6	7	
l_i	10	8	8	8	5	4	4	$\cdots$
$m(l_i)$	1	4	4	4	5	7	7	

Figure 4: An example illustrating eq. (2) with $l_i = \mathcal{C}_{\text{all}}^{(i)}$ for some suitable instance. Note that the lower bound is tight for $i \in \{1,5\}$ and the upper for $i = 2$.

As a direct consequence, we get for every $i \in \mathbb{N}$ that

$$i \;\leq\; m(\mathcal{C}_{\text{all}}^{(i)}) \;\leq\; i + \mathcal{C}_{\text{all}}(\mathcal{C}_{\text{all}}^{(i)}) - 1; \tag{2}$$

see Figure 4 for a sketch of the situation. Feasibility of $l := \mathcal{C}_{\text{all}}^{(k)}$ follows immediately. Now let $\hat{l} := \min\{l' \in \mathcal{C}_{\text{all}} \mid l' > l\}$; we see that

$$m(\hat{l}) \overset{(1)}{=} m(l) - \mathcal{C}_{\text{all}}(l) \overset{(2)}{\leq} k + \mathcal{C}_{\text{all}}(l) - 1 - \mathcal{C}_{\text{all}}(l) \;=\; k - 1$$

and therefore $\hat{l}$ is infeasible. By the choice of $\hat{l}$ and monotonicity of Feasible (cf. Lemma 3.1) we get that $l = \mathcal{C}_{\text{all}}^{(k)}$ is indeed the largest feasible candidate; this concludes the proof via Corollary 3.2 and Lemma 3.5. $\square$

Of course, we want to select from a small candidate set such as those we saw above; surely, selecting the kth largest element from these is not correct, in general. Also, not all restrictions may allow us to select because if we miss an L_i/j between two others, we may count wrong. The relation carries over to *admissible* restrictions with only small adaptions, though.

Corollary 4.5: *Let $\mathcal{C} = \mathcal{C}(I, f_l, f_u)$ be an admissible candidate multiset. Then,*

$$l^\star = \mathcal{C}^{(k')}$$

with $k' = k - \sum_{i \in I}[f_l(i) - 1]$.

Proof: With multiset

$$\mathbf{M} \;:=\; \biguplus_{i \in I} \{L_i/j \mid i \in I, j < f_l(i)\},$$

we get by Lemma 3.6 ii) and Lemma 3.1 that

$$\mathcal{C} \cap [l^\star, \infty) \;=\; (\mathcal{C}_{\text{all}} \cap [l^\star, \infty)) \setminus \mathbf{M}.$$

In addition, we know from Lemma 4.4 that

$$(\boldsymbol{\mathcal{C}}_{\text{all}} \cap [l^\star, \infty))^{(k)} \quad = \quad l^\star.$$

Since $\mathbf{M}$ contains only infeasible candidates (cf. Lemma 3.6 i) and Lemma 3.1), we also have that

$$\mathbf{M} \subset (l^\star, \infty),$$

and by definition

$$\mathbf{M} \cap \boldsymbol{\mathcal{C}} = \emptyset.$$

The claim

$$l^\star \quad = \quad \boldsymbol{\mathcal{C}}_{\text{all}}^{(k)} \quad = \quad \boldsymbol{\mathcal{C}}^{(k-|\mathbf{M}|)}$$

follows by counting. □

Hence, we can use any of the candidate sets we have investigated above. Instead of binary search we determine $l^\star$ by selecting the k'th largest element according to Corollary 4.5. Since selection takes only linear time we save a logarithmic factor compared to SearchLstar.

We give the full algorithm for completeness; note that steps 1 and 2 have not changed compared to SearchLstar.

Algorithm 3: SelectLstar$\langle f_l, f_u \rangle(\mathbf{L}, k)$:

```
1.  Compute $n' := \min(k, n+1)$.
2.  If $n' \leq n$:
    2.1. Determine $L_{\text{co}} := L^{(n')}$, i. e. the $n'$th largest length.
    2.2. If $L_{\text{co}}$ is optimal, answer $l^\star = L_{\text{co}}$ (and terminate).
2'. Otherwise (i. e. $n' > n$):
    2.4. Set $L_{\text{co}} := 0$.
3.  Assemble $I_{\text{co}} := I_{>L_{\text{co}}}$.
4.  Compute $\boldsymbol{\mathcal{C}} := \boldsymbol{\mathcal{C}}(I_{\text{co}}, f_l, f_u)$ as multiset.
5.  Determine $k' := k - \sum_{i \in I_{\text{co}}} [f_l(i) - 1]$.
6.  Answer $l^\star := \boldsymbol{\mathcal{C}}^{(k')}$.
```

Theorem 4.6:
Let $(I_{\text{co}}, f_l, f_u)$ be an admissible restriction where f_l and f_u can be evaluated in time $O(1)$.

Then, $\textsc{SelectLstar}\langle f_l, f_u\rangle$ *solves Envy-Free Stick Division in time and space $\Theta(n + |\mathcal{C}|)$.*

Proof: Correctness is clear from Lemma 5.2 and Corollary 4.5.

We borrow from the resource analysis of Theorem 4.1 with the following changes.

ad 4: We do not sort $\mathcal{C}$, so creating the multiset takes only time $\Theta(|\mathcal{C}|)$; the result takes up space $\Theta(|\mathcal{C}|)$, too, though.

ad 5,6: Instead of binary search on $\mathcal{C}$ with repeated evaluation of Feasible, we just have to compute k' (which clearly takes time $\Theta(|I_{\text{co}}|)$) and then select the k'th largest element from $\mathcal{C}$. This takes time $\Theta(|\mathcal{C}|)$ using e. g. the median-of-medians algorithm [Blu+73].

The resource requirements of the other steps remain unchanged, that is $\Theta(n)$. The bounds we claim in the corollary follow directly. □

Is has become clear now that decreasing the number of candidates is crucial for solving Envy-Free Stick Division quickly, provided we do not drop $l^\star$ along the way. We now endeavor to do so by choosing better admissible bounding functions.

5. Reducing the Number of Candidates

We can decrease the number of candidates significantly by observing the following. Whenever we cut $L^{(i)}$ (which is the ith largest length) into j pieces of length $L^{(i)}/j$ each, we also get at least j pieces of the same length from each of the longer sticks. In total, this makes for at least $i \cdot j$ pieces of length $L^{(i)}/j$; see Figure 5 for a visualization. By rearranging the inequality $k \geq i \cdot j$, we obtain a new admissible bound on j. For the algorithm, we have to sort I_{co}, though, so that $L_i = L^{(i)}$.

Example 2.1 Continued: For $\mathbf{L}_{\text{ex}}$ and $k = 9$, we get

$$\mathcal{C}(I_{\text{co}}, 1, \lceil k/i \rceil) = \left\{ \frac{8}{1}, \frac{8}{2}, \frac{8}{3}, \frac{8}{4}, \frac{8}{5}, \frac{8}{6}, \frac{8}{7}, \frac{8}{8}, \frac{8}{9},\ \frac{7}{1}, \frac{7}{2}, \frac{7}{3}, \frac{7}{4}, \frac{7}{5},\ \frac{6}{1}, \frac{6}{2}, \frac{6}{3} \right\},$$

that is 17 candidates (16 distinct ones); compare to $|\mathcal{C}(I_{\text{co}}, 1, k)| = 36$).

Lemma 5.1: *Assume that $L_{\text{co}} \neq l^\star$ and I_{co} is sorted w. r. t. decreasing lengths.*

Then, $\mathcal{C}(I_{\text{co}}, 1, \lceil k/i \rceil)$ is admissible.
Furthermore, $|\mathcal{C}(I_{\text{co}}, 1, \lceil k/i \rceil)| \in \Theta\big(k \cdot \log(\min(k, n))\big)$ in the worst case.

Proof: Again, we start by showing that $(I_{\text{co}}, 1, \lceil k/i \rceil)$ is an admissible restriction.

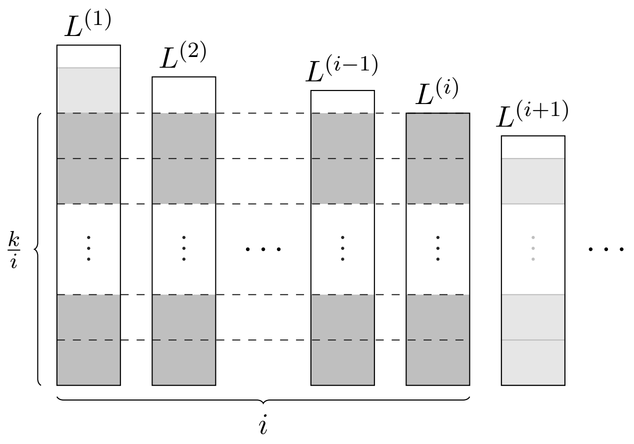

Figure 5: When considering cut lengths $L^{(i)}/j$, no j larger than $\lceil k/i \rceil$ is relevant. The sketch shows a cutting with $L^{(i)}$ and $j = k/i$. Note how we have k maximal pieces for sure (dark); there may be many more (light).

ad i), iii): Similar to the proof of Lemma 3.8.

ad ii): Because I_{co} is sorted, we have $L_i = L^{(i)}$ and $L_{i'} \geq L_i$ for $i' \leq i$. Therefore, we get for all the $l = L_i / f_u(i) = L_i \cdot \lceil k/i \rceil^{-1}$ with $i \in I_{\mathrm{co}}$ that

$$m(L) \;=\; \sum_{i'=1}^{n} \left\lfloor \frac{L_{i'}}{l} \right\rfloor \;\geq\; \sum_{i'=1}^{i} \left\lfloor \frac{L_i}{l} \right\rfloor \;=\; \sum_{i'=1}^{i} \left\lfloor \left\lceil \frac{k}{i} \right\rceil \right\rfloor \;\geq\; k.$$

This concludes the proof of the first claim.

For the size bound, let for short $\boldsymbol{\mathcal{C}} := \boldsymbol{\mathcal{C}}(I_{\mathrm{co}}, 1, \lceil k/i \rceil)$. Clearly, $|\boldsymbol{\mathcal{C}}| = \sum_{i \in I_{\mathrm{co}}} \lceil k/i \rceil$ (cf. Definition 3.4). With $|I_{\mathrm{co}}| = n' - 1 = \min(n, k-1)$ in the worst-case (cf. the proof of Lemma 3.8), the $\Theta(k \log n')$ bound on $|\boldsymbol{\mathcal{C}}|$ follows from

$$|\boldsymbol{\mathcal{C}}| \;=\; \sum_{i=1}^{n'-1} \left\lceil \frac{k}{i} \right\rceil \;\leq\; n' + \sum_{i=1}^{n'} \frac{k}{i} \;=\; n' + k \cdot H_{n'} \;\in\; \Theta(k \log n')$$

and

$$|\boldsymbol{\mathcal{C}}| \;=\; \sum_{i=1}^{n'-1} \left\lceil \frac{k}{i} \right\rceil \;\geq\; \sum_{i=1}^{n'-1} \frac{k}{i} \;=\; k \cdot H_{n'-1} \;\in\; \Theta(k \log n')$$

with the well-known asymptotic $H_k \sim \ln k$ of the harmonic numbers [GKP94, eq. (6.66)]. □

Combining Theorem 4.6 and Lemma 5.1 we have obtained an algorithm that takes time and space $\Theta(n + k \cdot \log(\min(k, n)))$. This is already quite efficient. By putting in some more work, however, we can save the last logarithmic factor that separates us from linear time and space.

5.1. Sandwich Bounds

Lemma 3.6 gives us some idea about what criteria we can use for restricting the set of lengths we investigate. We will now try to match these criteria as exactly as possible, deriving an interval $[\underline{l}, \bar{l}] \subseteq \mathbb{Q}_{>0}$ that includes $l^\star$ and is as small as possible; from these, we can infer almost as tight bounds (f_l, f_u).

Assume we have some length $L < l^\star$ and consider only lengths $l > L$. We have seen in Lemma 3.3 that we can then restrict ourselves to lengths from $I_{>L}$ when computing $m(l)$. Now, from the definition of m it is clear that we can sandwich $m(l)$ by

$$\sum_{i \in I_{>L}} \frac{L_i}{l} - 1 \;<\; m(l) \;\leq\; \sum_{i \in I_{>L}} \frac{L_i}{l}$$

for $l > L$. We denote for short $\Sigma_I := \sum_{i \in I} L_i$ for any $I \subseteq [1..n]$; rearranging terms, we can thus express these bounds more easily, both with respect to notational and computational effort. We get

$$\frac{\Sigma_{I_{>L}}}{l} - |I_{>L}| \;<\; m(l) \;\leq\; \frac{\Sigma_{I_{>L}}}{l} \tag{3}$$

for all $l > L$. Note that $L = 0$ is a valid choice, as then simply $I_{>L} = [1..n]$.

Rearranging these inequalities "around" $m(l) = k$ yields bounds on $l^\star$, which we can translate into bounds (f_l, f_u) on j (cf. Definition 3.4). We lose some precision because we round to integer bounds but that adds at most a linear number of candidates. A small technical hurdle is to ensure that both bounds are greater than our chosen L so that we can apply the sandwich bounds (3) in our proof.

Lemma 5.2: *Let L_{co} and I_{co} be defined as in Definition 3.7, and*

- $\underline{l} := \max\left\{L_{\text{co}}, \dfrac{\Sigma_{I_{\text{co}}}}{k + |I_{\text{co}}|}\right\}$ *and*
- $\bar{l} := \dfrac{\Sigma_{I_{\text{co}}}}{k}$.

Then, $\big(I_{\text{co}}, p(L_i, \bar{l}), p(L_i, \underline{l})\big)$ is admissible.

Proof: First, we determine what we know about our length bounds. Recall that $I_{\text{co}} = I_{>L_{\text{co}}} \neq \emptyset$ and L_{co} is not optimal.

We see that $\underline{l}$ is feasible by calculating

$$m(\underline{l}) \begin{cases} \overset{(3)}{>} \; \frac{\Sigma_{I_{\text{co}}}}{\underline{l}} - |I_{\text{co}}| \;=\; \frac{\Sigma_{I_{\text{co}}}}{\frac{\Sigma_{I_{\text{co}}}}{k+|I_{\text{co}}|}} - |I_{\text{co}}| \;=\; k, & \underline{l} > L_{\text{co}}, \\ = \; m(L_{\text{co}}) \;\geq\; k, & \underline{l} = L_{\text{co}} > 0, \end{cases} \tag{4}$$

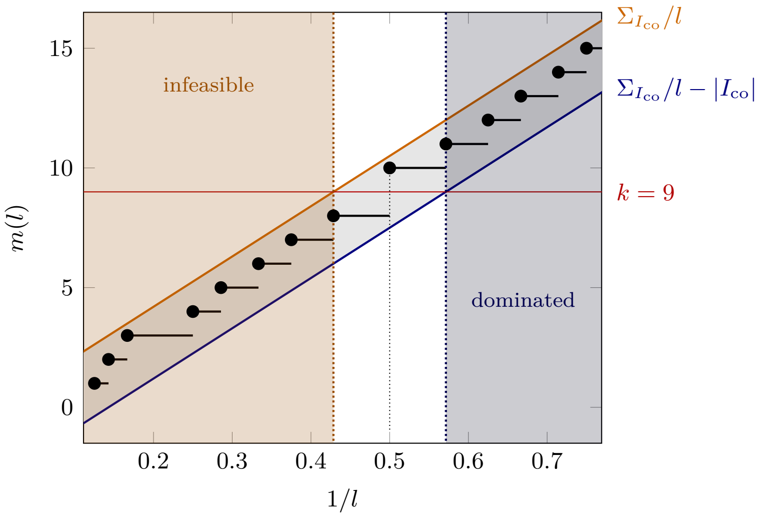

Figure 6: The number of maximal pieces $m(l)$ in the reciprocal of cut length l for $(\mathbf{L}_{\mathrm{ex}}, 9)$ as defined in in Example 2.1. Note how we can exclude all but three candidates (the filled circles) in a narrow corridor around $1/l^{\star} = 0.5$, defined by the points at which the bounds from (3) attain $k = 9$, namely $\underline{l} = 1.75$ and $\bar{l} = 2.\overline{3}$.

using in the second case that L_{co} is feasible. For the upper bound, we first note that because L_{co} is not optimal, there is some $\delta > 0$ with

$$\Sigma_{I_{\mathrm{co}}} \quad \geq \quad k(L_{\mathrm{co}} + \delta) \quad > \quad kL_{\mathrm{co}},$$

from which we get by rearranging that $\bar{l} > L_{\mathrm{co}}$. Therefore, we can bound

$$m(\bar{l} + \varepsilon) \quad \overset{(3)}{\leq} \quad \frac{\Sigma_{I_{\mathrm{co}}}}{\bar{l} + \varepsilon} \quad < \quad \frac{\Sigma_{I_{\mathrm{co}}}}{\bar{l}} \quad = \quad \frac{\Sigma_{I_{\mathrm{co}}}}{\frac{\Sigma_{I_{\mathrm{co}}}}{k}} \quad = \quad k \tag{5}$$

for any $\varepsilon > 0$, that is any length larger than $\bar{l}$ is infeasible. Note in particular that, in every case, $\bar{l} > \underline{l}$ so we always have a non-empty interval to work with.

We now show the conditions of Lemma 3.6 one by one.

ad i) Let $i \in I_{\mathrm{co}}$. If $p(L_i, \bar{l}) = 1$ the condition is trivially fulfilled. In the other case, we calculate

$$l \quad := \quad \frac{L_i}{p(L_i, \bar{l}) - 1} \quad = \quad \frac{L_i}{\lceil L_i/\bar{l} \rceil - 1} \quad > \quad \frac{L_i}{L_i/\bar{l}} \quad = \quad \bar{l}$$

and therewith $m(l) < k$ by (5).

ad ii) Let $i \in I_{\mathrm{co}}$ again. We calculate

$$l \quad := \quad \frac{L_i}{p(L_i, \underline{l})} \quad = \quad \frac{L_i}{\lceil L_i/\underline{l} \rceil} \quad \leq \quad \frac{L_i}{L_i/\underline{l}} \quad = \quad \underline{l}$$

which implies by Lemma 3.1 that

$$m(l) \;\geq\; m(\underline{l}) \;\overset{(4)}{\geq}\; k.$$

ad iii) See the proof of Lemma 3.8. □

Example 2.1 Continued: For $\mathbf{L}_{\text{ex}}$ and $k = 9$, we get

$$\boldsymbol{\mathcal{C}}\big(I_{\text{co}}, p(L_i, \bar{l}), p(L_i, \underline{l})\big) = \left\{\frac{8}{4}, \frac{8}{5}, \frac{7}{4}, \frac{7}{3}, \frac{6}{3}, \frac{6}{4}\right\},$$

that is six candidates (five distinct ones); compare to $|\boldsymbol{\mathcal{C}}(I_{\text{co}}, 1, \lceil k/i \rceil)| = 17$ and $|\boldsymbol{\mathcal{C}}(I_{\text{co}}, 1, k)| = 36$. See Figure 6 for a visualization of the effect our bounds have on the candidate set; note that we keep some additional candidates smaller than $\underline{l}$.

We see in this example that the bounds from Lemma 5.2 are not as tight as could be; $\boldsymbol{\mathcal{C}}(I_{\text{co}}, p(L_i, \bar{l}), p(L_i, \underline{l})) \cap [\underline{l}, \bar{l}]$ can be properly smaller (but not by more than one element per L_i), and since $l^\star \in [\underline{l}, \bar{l}]$ it is still a valid candidate set in some sense.

The reason for us sticking with the larger set is that we have defined admissibility in a way that is local to each L_i – we need to envelop $l^\star$ for each length – so we have no way to express *global* length bounds formally, at least not within this framework. In particular, using $f_u = i \mapsto \lfloor L_i/\underline{l} \rfloor$ is not admissible.

Nevertheless, we have obtained yet another admissible restriction and, as it turns out, it is good enough to achieve a linear candidate set. Only some combinatorics stand between us and our next corollary.

Lemma 5.3: $|\boldsymbol{\mathcal{C}}(I_{\text{co}}, p(L_i, \bar{l}), p(L_i, \underline{l}))| \in \Theta\big(\min(k, n)\big)$ *in the worst case.*

Proof: Recall that $|I_{\text{co}}| = \min(k-1, n)$ in the worst case (cf. the proof of Lemma 3.8). The upper bound on $|\boldsymbol{\mathcal{C}}|$ then follows from the following calculation:

$$\begin{aligned}
|\boldsymbol{\mathcal{C}}| &= \sum_{i \in I_{\text{co}}} \big[\, p(L_i, \underline{l}) - p(L_i, \bar{l}) + 1 \,\big] \\
&= |I_{\text{co}}| + \sum_{i \in I_{\text{co}}} \left\lceil \frac{L_i}{\underline{l}} \right\rceil - \sum_{i \in I_{\text{co}}} \left\lceil \frac{L_i}{\bar{l}} \right\rceil \\
&\leq |I_{\text{co}}| + \sum_{i \in I_{\text{co}}} \left[\frac{L_i}{\underline{l}} + 1 \right] - \sum_{i \in I_{\text{co}}} \frac{L_i}{\bar{l}} \\
&= |I_{\text{co}}| + \Sigma_{I_{\text{co}}} \cdot \frac{k + |I_{\text{co}}|}{\Sigma_{I_{\text{co}}}} + |I_{\text{co}}| - \Sigma_{I_{\text{co}}} \cdot \frac{k}{\Sigma_{I_{\text{co}}}} \\
&= 3 \cdot |I_{\text{co}}|.
\end{aligned}$$

A similar calculation shows the lower bound $|\boldsymbol{\mathcal{C}}| \geq |I_{\text{co}}|$. □

(f_l, f_u)	$\textsc{SearchLstar}\langle f_l, f_u\rangle$	$\textsc{SelectLstar}\langle f_l, f_u\rangle$
$(1, k)$	$\Theta(kn \log k)$	$\Theta(kn)$
$(1, \lceil k/i \rceil)^2$	$\Theta(k \log(k) \log(n))$	$\Theta(k \log n)$
$(p(L_i, \bar{l}), p(L_i, \underline{l}))$	$\Theta(n \log n)$	$\Theta(n)$

Figure 7: Assuming $k \geq n$, the table shows the worst-case runtime bounds shown above for the combinations of algorithm and bounding functions.

6. Conclusion

We have given a formal definition of Envy-Free Stick Division, derived means to restrict the search for an optimal solution to a small, discrete space of candidates, and developed algorithms that perform this search efficiently. Figure 7 summarizes the asymptotic runtimes of the combinations of candidate space and algorithm.

All in all, we have shown the following complexity bounds on our problem.

Corollary 6.1: *Envy-Free Stick Division can be solved in time and space $O(n)$.*

Proof: Algorithm $\textsc{SelectLstar}\langle p(L_i, \bar{l}), p(L_i, \underline{l})\rangle$ serves as a witness via Theorem 4.6, Lemma 5.2 and Lemma 5.3. □

A simple adversary argument shows that a sublinear algorithm is impossible; since the input is not sorted, adding a sufficiently large stick breaks any algorithm that does not consider all sticks. We have thus found an asymptotically optimal algorithm. Given the easy structure and almost elementary nature – we need but two calls to a selection algorithm – we expect it to be efficient in practice as well.

Acknowledgments

Erel Segal-Halevi[3] posed the original question [SH14] on Computer Science Stack Exchange. Our approach is based on observations in the answers by Abhishek Bansal (user1990169[4]), InstructedA[5] and FrankW[6]. Hence, even though the eventual algorithm and its presentation have been developed and refined offline with the use of a blackboard

[2] The additional time $\Theta(|I_{\text{co}}| \log |I_{\text{co}}|)$ necessary for sorting I_{co} as required by Lemma 5.1 is always dominated by generating $\boldsymbol{\mathcal{C}}$.

[3] `http://cs.stackexchange.com/users/1342/`

[4] `http://cs.stackexchange.com/users/19311/`

[5] `http://cs.stackexchange.com/users/20169/`

[6] `http://cs.stackexchange.com/users/13022/`

and lots of paper, the result has been the product of a small "crowd" collaboration made possible by the Stack Exchange platform.

We thank Chao Xu for pointing us towards the work by Cheng and Eppstein [CE14], and for providing the observation we utilize in Section 4.1.

References

[Blu+73] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. "Time Bounds for Selection." In: *Journal of Computer and System Sciences* 7.4 (Aug. 1973), pp. 448–461. DOI: 10.1016/S0022-0000(73)80033-9.

[BT96] Steven J. Brams and Alan D. Taylor. *Fair division.* Cambridge University Press, 1996. ISBN: 978-0-521-55644-6.

[CE14] Zhanpeng Cheng and David Eppstein. "Linear-time Algorithms for Proportional Apportionment." In: *International Symposium on Algorithms and Computation (ISAAC) 2014.* Springer, 2014. DOI: 10.1007/978-3-319-13075-0_46.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics: a foundation for computer science.* Addison-Wesley, 1994. ISBN: 978-0-20-155802-9.

[SH14] Erel Segal-Halevi. *Cutting equal sticks from different sticks.* Sept. 2014. URL: http://cs.stackexchange.com/q/30073 (visited on 11/26/2014).

[SHHA15] Erel Segal-Halevi, Avinatan Hassidim, and Yonatan Aumann. "Waste Makes Haste: Bounded Time Protocols for Envy-Free Cake Cutting with Free Disposal." In: *The 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* May 2015.

A. Notation Index

In this section, we collect the notation used in this paper. Some might be seen as "standard", but we think including them here hurts less than a potential misunderstanding caused by omitting them.

Generic Mathematical Notation

$[1..n]$	The set $\{1,\ldots,n\} \subseteq \mathbb{N}$.
$\lfloor x \rfloor$, $\lceil x \rceil$	floor and ceiling functions, as used in [GKP94].
$\ln n$	natural logarithm.
$\log^2 n$	$(\log n)^2$
H_n	nth harmonic number; $H_n = \sum_{i=1}^{n} 1/i$.
p_n	nth prime number.
$\mathbf{A}$	multisets are denoted by bold capital letters.
$\mathbf{A}(x)$	multiplicity of x in $\mathbf{A}$, i. e., we are using the function notation of multisets here.
$\mathbf{A} \uplus \mathbf{B}$	multiset union; multiplicities add up.
$\mathbf{L}^{(k)}$	The kth largest element of multiset $\mathbf{L}$ (assuming it exists); if the elements of $\mathbf{L}$ can be written in non-increasing order, $\mathbf{L}$ is given by $\mathbf{L}^{(1)} \geq \mathbf{L}^{(2)} \geq \mathbf{L}^{(3)} \geq \cdots$. *Example:* For $\mathbf{L} = \{10, 10, 8, 8, 8, 5\}$, we have $\mathbf{L}^{(1)} = \mathbf{L}^{(2)} = 10$, $\mathbf{L}^{(3)} = \mathbf{L}^{(4)} = \mathbf{L}^{(5)} = 8$ and $\mathbf{L}^{(6)} = 5$.

Notation Specific to the Problem

stick	one of the lengths of the input, before any cutting.
piece	one of the lengths after cutting; each piece results from one input stick after some cutting operations.
maximal piece	piece of maximal length (after cutting).
n	number of sticks in the input.
$\mathbf{L}$, L_i, L	$\mathbf{L} = \{L_1, \ldots, L_n\}$ with $L_i \in \mathbb{Q}_{>0}$ for all $i \in [1..n]$ contains the lengths of the sticks in the input. We use L as a free variable that represents (bounds on) input stick lengths.
k	$k \in \mathbb{N}$, the number of maximal pieces required.

l	free variable that represents (bounds on) candidate cut lengths; by Lemma 3.1 only $l = {}^{L_i}/_{j}$ for $j \in \mathbb{N}$ have to be considered.
$l^\star$	the optimal cut length, i. e., the cut length that yields at least k maximal pieces while minimizing the total length of non-maximal (i. e. waste) pieces.
$c(L, l)$	the number of cuts needed to cut stick L into pieces of lengths $\leq l$; $c(L, l) = \lceil \frac{L}{l} - 1 \rceil$.
$m(L, l)$	the number of maximal pieces obtainable by cutting stick L into pieces of lengths $\leq l$; $m(L, l) = \lfloor \frac{L}{l} \rfloor$.
$p(L, l)$	the minimal total number of pieces resulting from cutting stick L into pieces of lengths $\leq l$; $p(L, l) = \lceil \frac{L}{l} \rceil$.
$c(l) = c(\mathbf{L}, l)$	total number of cuts needed to cut all sticks into pieces of lengths $\leq l$; $c(\mathbf{L}, l) = \sum_{L \in \mathbf{L}} c(L, l)$.
$m(l) = m(\mathbf{L}, l)$	total number of maximal pieces resulting from cutting stick L into pieces of lengths $\leq l$; $m(\mathbf{L}, l) = \sum_{L \in \mathbf{L}} m(L, l)$.
$\text{Feasible}(l) = \text{Feasible}(\mathbf{L}, k, l)$	indicator function that is 1 when l is a feasible length and 0 otherwise; $\text{Feasible}(\mathbf{L}, k, l) = [m(\mathbf{L}, l) \geq k]$.
$\boldsymbol{\mathcal{C}}(I, f_l, f_u)$	multiset of candidate lengths ${}^{L_i}/_{j}$, restricted by the index set I of considered input sticks L_i, and lower resp. upper bound on j; cf. Definition 3.4 (page 13).
$\boldsymbol{\mathcal{C}}_{\text{all}}$	The unrestricted (infinite) candidate set $\boldsymbol{\mathcal{C}}_{\text{all}} = \boldsymbol{\mathcal{C}}([1..n], 1, \infty)$.
admissible restriction (I, f_l, f_u)	sufficient conditions on restriction (I, f_l, f_u) to ensure that Envy-Free Stick Division $\in \mathcal{C}(I, f_l, f_u)$; cf. Lemma 3.6 (page 13).
$I_{>L}$	the set of indices of input sticks $L_i > L$; cf. Lemma 3.3.
I_{co}, L_{co}	$I_{\text{co}} = I_{>L_{\text{co}}}$ is our canonical index set with cutoff length L_{co} the kth largest input length; cf. Definition 3.7 (page 14).
Σ_I	assuming $I \subseteq [1..n]$, this is a shorthand for $\sum_{i \in I} L_i$.
$\underline{l}$, $\bar{l}$	lower and upper bounds on candidate lengths $\underline{l} \leq l \leq \bar{l}$ so that $l^\star \in \mathcal{C}_{\text{all}} \cap [\underline{l}, \bar{l}]$; see Lemma 5.2 (page 23).

B. On the Number of Distinct Candidates

As mentioned in Section 4, algorithm SEARCHLSTAR can profit from removing duplicates from the candidate multisets during sorting. We will show in the subsequent proofs that none of the restrictions introduced above cause more than a constant fraction of all candidates to be duplicates.

We denote with $\mathcal{C}(\dots)$ the set obtained by removing duplicates from the multiset $\boldsymbol{\mathcal{C}}(\dots)$ with the same restrictions.

Lemma B.1: $|\mathcal{C}(I_{\mathrm{co}}, 1, k)| \in \Theta(|\boldsymbol{\mathcal{C}}(I_{\mathrm{co}}, 1, k)|)$ *in the worst case.*

Proof: Let for short $C := |\mathcal{C}(I_{\mathrm{co}}, 1, k)|$ and $\boldsymbol{\mathcal{C}} := \boldsymbol{\mathcal{C}}(I_{\mathrm{co}}, 1, k)$. It is clear that $C \leq |\boldsymbol{\mathcal{C}}|$; we will show now that $C \in \Omega(|\boldsymbol{\mathcal{C}}|)$ in the worst case.

Consider instance

$$\mathbf{L}_{\text{primes}} = \{p_n, \dots, p_1\}$$

with p_i the ith prime number and any $k \in \mathbb{N}$; note that $L_i = p_{n-i+1}$. Let for ease of notation $n' := \min(k, n+1)$; note that $L_{\mathrm{co}} = \mathbf{L}_{\text{primes}}^{(n')}$ if $k \leq n$. We have $|I_{\mathrm{co}}| = \min(k-1, n)$ because the L_i are pairwise distinct, and therefore $|\boldsymbol{\mathcal{C}}| = k|I_{\mathrm{co}}| = k(n'-1)$. Since the L_i are also pairwise coprime, all candidates L_i/j for which j is *not* a multiple of L_i are pairwise distinct. Therefore, we get

$$\begin{aligned}
C &\geq |\boldsymbol{\mathcal{C}}| - \sum_{i=n-n'+2}^{n} \left\lfloor \frac{k}{p_i} \right\rfloor \\
&\geq |\boldsymbol{\mathcal{C}}| - \sum_{i=n-n'+2}^{n} \frac{k}{p_i} \\
&= |\boldsymbol{\mathcal{C}}| - (n'-1)k \cdot \sum_{i=n-n'+2}^{n} \frac{1}{p_i} \\
&\geq |\boldsymbol{\mathcal{C}}| - |\boldsymbol{\mathcal{C}}| \cdot \frac{n'}{p_{n'}} \\
&\geq |\boldsymbol{\mathcal{C}}| - |\boldsymbol{\mathcal{C}}| \cdot \frac{2}{3} \\
&= \frac{|\boldsymbol{\mathcal{C}}|}{3}.
\end{aligned}$$

In particular, we can show that $k/p_k \leq 2/3$ by $k/p_k < 0.4$ for $k \geq 20$ [GKP94, eq. (4.20)] and checking all $k < 20$ manually; the maximum is attained at $k = 2$. □

Lemma B.2: $|\mathcal{C}(I_{\mathrm{co}}, 1, \lceil k/i \rceil)| \in \Theta(|\boldsymbol{\mathcal{C}}(I_{\mathrm{co}}, 1, \lceil k/i \rceil)|)$ *in the worst case.*

Proof: Let for short $C := |\mathcal{C}(I_{\mathrm{co}}, 1, \lceil k/i \rceil)|$ and $\boldsymbol{\mathcal{C}} := \boldsymbol{\mathcal{C}}(I_{\mathrm{co}}, 1, \lceil k/i \rceil)$. It is clear that $C \leq |\boldsymbol{\mathcal{C}}|$; we will show now that $C \in \Omega(|\boldsymbol{\mathcal{C}}|)$ in the worst case.

We make use of the same instance $(\mathbf{L}_{\text{primes}}, k)$ we used in the proof of Lemma B.1, with a similar calculation:

$$\begin{aligned}
C &= |\boldsymbol{\mathcal{C}}| - \sum_{i=n-n'+2}^{n} \left\lfloor \frac{\lceil k/i \rceil}{p_i} \right\rfloor \\
&\geq |\boldsymbol{\mathcal{C}}| - \sum_{i=n-n'+2}^{n} \frac{\frac{k}{i}+1}{p_i} \\
&\geq |\boldsymbol{\mathcal{C}}| - \frac{n'-1}{p_{n'-1}} \cdot \left(1 + \frac{k}{n'-1}\right) \\
&\geq |\boldsymbol{\mathcal{C}}| - \frac{2}{3} \cdot \left(1 + \frac{k}{n'-1}\right) \\
&\in \Theta(|\boldsymbol{\mathcal{C}}|)
\end{aligned}$$

because $k/n' \in o(k \log n') = o(|\boldsymbol{\mathcal{C}}|)$. □

Lemma B.3: $|\mathcal{C}(I_{\text{co}}, p(L_i, \bar{l}), p(L_i, \underline{l}))| \in \Theta(|\boldsymbol{\mathcal{C}}(I_{\text{co}}, p(L_i, \bar{l}), p(L_i, \underline{l}))|)$ *in the worst case.*

Proof: Let again for short $C := |\mathcal{C}(I_{\text{co}}, p(L_i, \bar{l}), p(L_i, \underline{l}))|$ and $\boldsymbol{\mathcal{C}} := \boldsymbol{\mathcal{C}}(I_{\text{co}}, p(L_i, \bar{l}), p(L_i, \underline{l}))$. It is clear that $C \leq |\boldsymbol{\mathcal{C}}|$; we will show now that $C \in \Omega(|\boldsymbol{\mathcal{C}}|)$ in the worst case.

We make use of our trusted instance $(\mathbf{L}_{\text{primes}}, k)$ again. We show that very prime yields at least one candidate unique to itself, as long as k is constant (which is sufficient for a worst-case argument).

Recall that $\bar{l} > \underline{l}$ so every L_i *has* some j; we note furthermore that for fixed $i \in I_{\text{co}}$,

$$j \;\leq\; p(L_i, \underline{l}) \;=\; \left\lceil L_i \Big/ \frac{\Sigma_{I_{\text{co}}}}{k + |I_{\text{co}}|} \right\rceil \;=\; \left\lceil p_i \cdot \frac{k + |I_{\text{co}}|}{\sum_{i' \in I_{\text{co}}} p_{i'}} \right\rceil \;\leq\; \left\lceil p_i \cdot \frac{2k}{p_n} \right\rceil \;\leq\; 2k \;<\; L_i$$

for big enough n, in particular because $p_n \sim n \ln n$ [GKP94, p 110]. That is, every L_i with $i \in I_{\text{co}}$ yields at least one L_i/j no other does, since all L_i are co-prime. Hence $C \geq |I_{\text{co}}| \in \Theta(|\boldsymbol{\mathcal{C}}|)$. □